

# Using Two AP2 Cards with S232 Drivers TDT Tech Note #156

---

## Overview

This technote describes how to use 2 AP2 cards with the S232 drivers. These instructions replace the `ap_init()` and `ap_select()` commands from the 16-bit APOS drivers.

## Working With Dual Card Systems

System II can support up to two AP2 cards simultaneously. It can be advantageous to have dual cards as the computational resources for that system can be effectively doubled. Each card is a *logical device*.

## What Are Logical Devices?

Logical devices can be thought of as physical hardware, i.e. AP2 cards. An AP2 card can exist as one of two varieties, APa or APb. If only one AP2 card is present in a given system, that card is by default Logical Device 0 independent of the memory area in which the card resides. If it is a dual card system, either card, APa or APb, can be defined as Logical Device 0. By default, the remaining card is Logical Device 1.

## Connecting to Logical Devices

Each logical device, and hence AP2 card, requires an independent device driver. For Windows 95/98, the device drivers are `WRTdev0.VxD` and `WRTdev1.VxD`. For Windows NT, the device driver is `WinRT.sys`. Each have their own settings stored in the Windows Registry which are loaded at the time Windows is started. In addition to a device driver, each AP2 card must be associated with an S232 Controller which provides a core set of services. Those services include transferring the AP2 operating system to the AP2 card.

In Windows, applications can be launched more than once. Once launched, each copy of the program functions "independently" of the others. For example, the Windows provided program Notepad can be executed several times with each copy opening and editing its own document. In a similar fashion, dual copies of the S232 Controller program can be run simultaneously with each copy connected to a different device driver/AP2 card pair.

The S232 Controller requires a command line argument to connect to a specific device driver/AP2 card pair. A command line argument follows the executables name and includes the switch identifier, `"/`. When the S232 Controller program is launched, the command line argument tells the application which logical device to use, e.g. `/1` is logical device 1. The absence of a command line arguments implies logical device 0.

## Setting the Command Line Argument

To add a command line argument to the S232 Controller:

1. Find the S232 Controller shortcut icon, *not the program itself*, by using the Windows Explorer.
2. Select the S232 Controller shortcut.
3. Click the right mouse button.
4. Select Properties.
5. Find the Target Window that contains the path to the S232 Controller application: the default is C:\Tdt\S232\S2ctrl32.exe.
6. Add the required command-line argument, e.g. /1. The entire line now appears in the default case as: C:\Tdt\S232\S2ctrl32.exe /1.
7. Click OK.

## A New Look For A Second Card

This application *is* the S232 Controller and contains all the functionality that has been previously described. It can perform diagnostic tests, communicate with the System II modules on its own XBUS, examine buffers either on the stack or in DAMA, and work with System II 32-bit applications.



The only difference is what is present on the title bar and in the notification tray of the taskbar. The title bar contains the name of the logical device that it controls, e.g. S232 Controller (Dev1). The S232 Controller that communicates with logical device 1 includes a yellow Roman numeral as shown to the left.

This S232 Controller icon functions exactly as previously described.

## Function Calls for Initializing and Switching Between Cards

### Overview

`S2init()` is called to initialize each AP2 card (device numbers 0 and 1). One important difference here is that you need to create a local copy of the data structure that is created when each AP2 is initialized. This is accomplished through the `getS2link()` which returns the memory location of the AP2 data structure in the driver. This memory area from the driver is copied to the local memory area. The `setS2link()` function is then used to select the AP2 card to use by changing the pointer to the appropriate local copy of the AP2 data structure.

## S2init(dn, mode, apt)

<b>Prototype</b>	int S2init(int dn, int mode, int apt);
<b>Operation</b>	Initializes the specified AP2 and identifies the XBUS modules present in the IDed caddies. The apt value defines the global timeout for the application. Should any System II call take longer than the timeout, an error may result. S2init() supercedes apinit() and Xbinit().
<b>Arguments</b>	<i>dn</i> selects the logical device number <i>mode</i> selects the application mode INIT_PRIMARY 1 INIT_SECONDARY 2 INIT_EITHER 3 INIT_FORCEPRIM 4 INIT_OP2 5 INIT_OP2RESET 6 <i>apt</i> selects the global system timeout in ms
<b>Returns</b>	The mode value of the application successfully initialized or 0 if unsuccessful.
<b>Example</b>	The following code initializes the AP2 and XBUS in a Primary application. The global timeout is set to 5000 ms. If the initialization is unsuccessful, a dialog window will display an error message.

```
if(!S2init(0, INIT_PRIMARY,  
5000))  
{  
    AfxMessageBox("Unable to  
initialize a primary app.");  
    exit(0);  
}
```

## getS2link()

**Prototype** int getS2link(void);

**Operation** Returns pointer to TDT driver data structure for AP2 card. This is used after S2init() to obtain the data address for the AP2 data structure. This data structure is then copied into a local data structure (otherwise it would be copied over when S2init() was run again with a different logical device number for another AP2 card).

**Arguments** *dn* selects the logical device number

**Example** This example initializes two AP2 cards and then copies over the AP2 memory structures to two local arrays: AP0 and AP1.

```
char AP0[10000], AP1[10000];
if(!S2init(0, INIT_SECONDARY, 5000))
{
    cout << "Cannot initialize device 0" << endl;
    return;
}
memcpy(AP0, (void*)getS2link(), sizeof(AP0));

if(!S2init(1, INIT_SECONDARY, 5000))
{
    cout << "Cannot initialize device 1" << endl;
    return;
}
memcpy(AP1, (void*)getS2link(), sizeof(AP1));
```

## setS2link(lp)

**Prototype** void setS2link(int lp);

**Operation** Sets pointer to data structure for AP2 that you want to select.

**Arguments** *lp* pointer

**Example** In this example, the first AP2 (logical device zero) is selected. Subsequent AP2 commands will be carried out on this card. AP0 is the local data structure that was initialized in the example under getS2link().

```
setS2link(AP0);
```

## Example Code

The example MSVC++ code (DualCards.cpp) that accompanies this technote shows how to use two AP2 cards in one program. This code is included below.

```
#include <windows.h>
#include "s232.h"
#include <iostream.h>

// Function Calls
void ToDevice0();
void ToDevice1();

// Global variables
char AP0[DRIVERSIZE], AP1[DRVIERSIZE];

void main()
{
    int DeviceStack, i;
    float ToneFreq;
    int LoopFlag = 0;
    char emess[255];

    if(!S2init(0, INIT_SECONDARY, 5000))
    {
        cout << "Cannot initialize device 0" << endl;
        return;
    }
    memcpy(AP0, (void*)getS2link(), sizeof(AP0));

    if(!S2init(1, INIT_SECONDARY, 5000))
    {
        cout << "Cannot initialize device 1" << endl;
        return;
    }
    memcpy(AP1, getS2link(), sizeof(AP1));

    do
    {
        cout << "Push a tone to which logical card (0 or 1)
? " ;
        cin >> DeviceStack;
        cout << "What frequency (Hz)? " ;
        cin >> ToneFreq;

        if(DeviceStack)
```

```

        ToDevice1();
    else
        ToDevice0();

    if(APlock(100, 0))
    {
        dpush(4096);
        tone(ToneFreq, 20.0);
        APunlock(0);
    }
    else
    {
        getS2err(emess);
        cout << emess << endl;
    }

    cout << endl << "Enter 1 to continue or 0 to quit.
";
    cin >> LoopFlag;
}while(LoopFlag);

// Remove all buffers from the cards
for(i = 0; i < 2; i++)
{
    if(i)
        ToDevice1();
    else
        ToDevice0();
    if(APlock(100, 0))
    {
        dropall();
        APunlock(0);
        S2close();
    }
    else
    {
        getS2err(emess);
        cout << emess << endl;
    }
}
}

void ToDevice0()
{
// Save any additional state information here
    setS2link(AP0);
}

```

```
void ToDevice1()  
{  
  // Save any additional state information here  
  setS2link(AP1);  
}
```

## BrainWare Example

BrainWare requires some code in addition to that in the above example, because the Stimulus DLL is going to be exited before BrainWare exits. Thus, any local memory structures created in the Stimulus DLL will not exist when the DLL is closed. When the stimulus DLL exits, it must copy the original AP2 data structure from the local structure (AP0 in the example) back to the original structure, and it must point to this structure.

The two DLLs, AP2dev0.dll and AP2dev1.dll use two AP2 cards to create and play different tones out of two separate D/A converters. The AP2dev0.dll should be loaded into BrainWare first, and then AP2dev1.dll should be loaded. Pay attention to the code in InitializeStimulusDLL() and tidyUpStimulusDLL() in the AP2dev1.dll.

Some important points:

- Use only one DAparameters() statement. Otherwise BrainWare will give you an error because it does not know you are using two cards.
- The last DLL loaded should control the triggering of both sets of hardware.